

SECONDO ASSEGNAMENTO: BBGAME

Informatica, Corso di Laurea in Fisica, Università di Pisa

AA 2019/20

1 Bubble Breaker Game

Il gioco Bubble Breaker si basa su una griglia, rappresentata da una matrice di caratteri di dimensione $L \times L$, dove ogni cella contiene un carattere (o colore), oppure è *vuota*.

Vi sono h colori in totale (è un input del gioco), rappresentati dai numeri $0, 1, \dots, h-1$, che corrispondono alle celle *piene*.

Ad esempio, se $h = 3$, i possibili colori corrispondono ai numeri 0, 1, e 2. Il valore delle celle vuote è definito dalla macro EMPTY in `bbgame.h`.

Le regole del gioco sono le seguenti:

- La board del gioco è una matrice $L \times L$ di numeri *interi*, dove la cella `board[0][0]` corrisponde alla cella in alto a sinistra.
- I parametri della partita sono n, h, k : La board avrà dimensione $n \times n$, e le celle h possibili valori compresi tra 0 e $h-1$. Il giocatore ha a disposizione k token di riempimento.
- All'inizio della partita la board viene inizializzata con valori casuali.
- Il giocatore può effettuare una mossa selezionando una cella i, j , o usare un token di riempimento per riempire tutte le celle vuote (sempre in modo casuale tra 0 e $h-1$), se ha ancora token a disposizione.
- Le mosse sono solo di tipo "bubble" (vedi descrizione delle funzioni), e sono valide solo se la bubble è composta da almeno due celle.
- La partita termina quando non ci sono mosse valide disponibili e il giocatore non può riempire ulteriormente la board (ad es, perché ha finito i token). Il punteggio della partita è la somma dei punteggi di tutte le mosse valide eseguite dal giocatore.
- Nel momento in cui termina la partita, il gioco chiede un nome al giocatore e aggiorna il suo punteggio nel file di testo "highscores.txt" (vedi descrizione delle funzioni).

2 Regole di pop bubble

L'unica regola di pop è `pop_bubble`, da realizzare in `int pop_bubble (partita_t * p, int i, int j)`.

Data una cella i, j *non vuota*, contenente il valore v , la *bubble* di i, j corrisponde alla cella `board[i][j]` e tutta la porzione raggiungibile dalla cella i, j della matrice contenente lo stesso carattere.

Ogni cella è considerata vicina alla cella alla sua sinistra, destra, sopra, e sotto, ma non diagonalmente. La figura sotto fa vedere un esempio.

Una mossa è considerata valida se la bubble è composta da *almeno* due celle. Non è ammesso colpire bubble composte da una sola cella, o celle vuote.

Se la mossa è valida, `pop_bubble` esplode tutte le celle nella bubble di i, j , e applica le regole di gravità avanzate, che fanno cadere le celle piene verso il basso e compattano le colonne vuote facendo scorrere quelle non vuote verso destra (vedi descrizione delle funzioni).

La funzione restituisce il punteggio assegnato alla mossa c^2 , dove c è il numero di celle esplose dalla mossa; in caso di mossa non valida (cella vuota, bubble di dimensione 1, o coordinate non valide), la funzione restituisce il valore ERRORE (definito in `bbgame.h`), e non modifica la board. Vediamo un esempio:

0 0 0 0 0 2	0 0 0 0 0 2	0 0 0 2	* * * 2
1 1 0 2 0 0	* * 0 2 0 0	0 2 0 0	(*)2 * *
1 1 0 1 0 1	*(*)0 1 0 1	0 0 0 1 0 1	* * * 1 * 1
2 2 0 2 0 2	2 2 0 2 0 2	2 2 0 2 0 2	2 2 * 2 * 2
1 0 0 2 0 3	1 0 0 2 0 3	1 0 0 2 0 3	1 * * 2 * 3
3 2 0 2 2 0	3 2 0 2 2 0	3 2 0 2 2 0	3 2 * 2 2 0
La board 6x6 di una partita p	*: celle esplose da <code>pop_bubble(p, 2, 1)</code>	La board dopo <code>pop_bubble(p, 2, 1)</code>	*: celle esplose da <code>pop_bubble(p, 1, 2)</code>
	2 2	2 2	2
	1 1	1 1	1
	2 2 2	2 * 2	2 2
	1 2 2 3	1 * * 3	1 2 3
	3 2 2 2 0	3(*)* * 0	3 1 0
	La board dopo <code>pop_bubble(p, 1, 2)</code>	*: celle esplose da <code>pop_bubble(p, 5, 2)</code>	La board dopo <code>pop_bubble(p, 5, 2)</code>

Notare in particolare come le colonne vuote sono compattate spostando le piene verso destra, e come alla fine non ci sono mosse valide disponibili in quanto tutte le bubble rimaste sono da 1 cella: in questo caso se il giocatore non ha token di riempimento a disposizione, la partita termina.

3 Aggiornamento del punteggio

Il file “`highscores.txt`” contiene la lista dei nomi dei giocatori che hanno giocato, con il loro punteggio migliore, separati da uno spazio. L’ordine in cui le righe appaiono nel file è irrilevante. Ad esempio:

```
Giuseppe 50
Matteo 49
```

Il file può essere aggiornato con la funzione `int aggiorna_punteggio (char * nomefile, char * nome, int punteggio)`. Ad esempio, supponendo il file sia come sopra, chiamando `aggiorna_punteggio ('highscores.txt', 'Sergio', 70)`, il file deve contenere, in qualsiasi ordine, le righe:

```
Giuseppe 50
Matteo 49
Sergio 70
```

In quanto non c’era alcun punteggio per il giocatore “Sergio”. Ci aspettiamo che la funzione restituisca 1 in quanto il punteggio è stato registrato.

Se chiamiamo `aggiorna_punteggio ('highscores.txt', 'Giuseppe', 55)`, ci aspettiamo:

```
Matteo 49
Sergio 70
Giuseppe 55
```

Anche qui ci aspettiamo 1 come valore restituito. Tuttavia, se chiamiamo `aggiorna_punteggio ('highscores.txt', 'Matteo', 35)`, il giocatore “Matteo” ha già un punteggio più elevato, quindi ci aspettiamo che non venga fatta alcuna modifica al file e venga restituito il valore 0.

4 Cosa deve essere realizzato

Lo studente deve realizzare un file `bbgame.c` contenente le funzioni che servono ad eseguire i componenti del gioco BUBBLE BREAKER, inclusa una funzione `gioca` che permette al giocatore di giocare una partita.

I prototipi delle funzioni da realizzare in `bbgame.c` si trovano nel file `bbgame.h`.

- `partita_t * crea_partita(int n, int h, int k)`: crea una nuova partita, allocando dinamicamente la memoria e inizializzandone tutti i campi opportuni, inclusa la board, e inizializza ogni sua cella con un intero nell'intervallo $0, 1, \dots, h - 1$ scelto in modo casuale. Restituisce la partita creata.
- `void distruggi_partita(partita_t * p)`: distrugge la partita data, liberando la memoria in modo appropriato.
- `void gravita (partita_t * p)`: applica le regole di gravità sulla board della partita data. Fa cadere gli elementi in celle che si trovano sopra a celle vuote verso il basso, fino ad appoggiarsi sulle celle non vuote sottostanti, o sul fondo. Successivamente compatta le colonne facendo scorrere verso destra le colonne non vuote finché possibile.
- `int pop_bubble (partita_t * p, int i, int j)`: esegue la mossa sulla cella i, j della partita p . In particolare esplode la cella i, j e tutta la sua bubble, come in figura sopra, se questa è composta da almeno due celle. Poi applica le regole di gravità. La funzione restituisce il punteggio della mossa (ovvero il quadrato del numero di celle esplose), o `ERRORE` in caso di mossa invalida (ad es, colpisco una cella vuota, o fuori dalla board, o una bubble da una sola cella). In caso di errore non viene fatta alcuna modifica alla partita.
- `int riempi (partita_t * p)`: Riempie tutte e sole le celle vuote nella board della partita p , usando la soglia h contenuta in essa (e quindi valori casuali nell'intervallo $0, 1 \dots h - 1$). Restituisce 1 se la board è stata modificata, 0 altrimenti.
- `int valida (partita_t * p)`: restituisce 1 se c'è almeno una mossa valida che può essere eseguita nella board della partita p , 0 altrimenti.
- `int aggiorna_punteggio (char * nomefile, char * nome, int punteggio)`: Aggiorna il file `nomefile` con il nuovo punteggio, come descritto sopra, creando il file se non esiste. Restituisce 1 se il file è stato modificato, 0 altrimenti (ad es, se il giocatore aveva già un punteggio superiore).
- `int gioca (int n, int h, int k)`: Gestisce un'intera partita di *bubble breaker* tramite le regole spiegate sopra, interfacciandosi con l'utente tramite il terminale. A fine partita chiede un nome al giocatore ed aggiorna opportunamente il file "highscores.txt" col punteggio. Restituisce 0 se il programma termina correttamente, 1 in caso di terminazione precoce (ad es, impossibile creare la partita).

I file `*test.c` contengono dei `main` che usano queste funzioni ed effettuano dei test sul loro funzionamento. Tali test possono essere attivati automaticamente utilizzando il `Makefile` come specificato nel file `README`. Solo il codice che supera con successo questi test può essere consegnato.

Tuttavia, è bene ricordare che il superamento dei test non garantisce la correttezza completa della soluzione, quindi invitiamo gli studenti ad analizzare attentamente i risultati ottenuti e le stampe effettuate prima della consegna.

5 Consegna

Lo studente deve consegnare un archivio contenente i file `bbgame.c` e `gruppo.txt` (le istruzioni su come generare questo archivio sono nel file `README`).

Le funzioni realizzate devono essere adeguatamente commentate come discusso a lezione.

Funzioni ausiliarie e materiale aggiuntivo: E' chiaramente possibile realizzare funzioni ausiliarie nel file `bbgame.c` (ad esempio, per la stampa della board). E' anche possibile, se si vuole, realizzare funzionalità aggiuntive e consegnare una breve discussione su quanto realizzato e come utilizzarlo sotto forma di file PDF. Questo però non deve modificare il normale funzionamento delle funzioni richieste dalla consegna (il codice consegnato deve in ogni caso passare i test).